Simulation and Analysis of Different Switch Architectures for Interconnection Networks in MIMD Shared Memory Machines

by

Yue-Sheng Liu and Susan Dickey

Ultracomputer Note # 141
June 1988

Ultracomputer Research Laboratory

New York University
Courant Institute of Mathematical Sciences
Division of Computer Science
251 Mercer Street, New York, NY 10012

[redacted]

Simulation and Analysis of Different Switch Architectures for Interconnection Networks in MIMD Shared Memory Machines

by

Yue-Sheng Liu and Susan Dickey

Ultracomputer Note # 141
June 1988

This work was carried out as part of the research program on parallel processing at the Ultracomputer Research Laboratory of the Courant Institute of Mathematical Sciences. The program is supported in part under a Joint Study Agreement with IBM, by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under contract number DE-AC02 76ER03077, and by the National Science Foundation, under grant number DCR-8413359. The views and opinions expressed herein do not necessarily reflect those of IBM or of the United States Government or any agency thereof.

ABSTRACT

Differences in switch architectures can have a significant effect on both latency and throughput in interconnection networks, and thus can affect overall performance in shared memory parallel systems with processor to memory interconnection networks. In this paper we describe several switch architectures which differ in the presence or absence of buffers, in the location and functionality of the buffers, and in the width of the data path.

Since the addition of an architectural feature may have a significant cost in the complexity or cycle speed of a switching component, it is important to have an accurate assessment of the benefit provided. Therefore, we make quantitative comparisons of switch performance using both simulation and analytical methods.

1. Introduction

The basic system model under consideration is illustrated in Figure 1. N independent processing elements (PEs) are connected to M shared memory modules (MMs) through a logarithmic interconnection network. Examples of this architecture include the NYU Ultracomputer [4], the BBN Butterfly [16], and the IBM RP3 [15].

In this model, traffic through the network consists of requests from a PE to an MM and responses from an MM to a PE. Requests and responses are typically sent as messages of at most a few hundred bits, divided into packets of the same size as the data path width from switch to switch within the network. Packets within the same message are pipelined: when the first packet is accepted by a switch, it is forwarded to the next switch the next cycle, if there is no queue and the output port is not blocked by the next stage. Remaining packets of an accepted message follow on succeeding cycles without interruption.

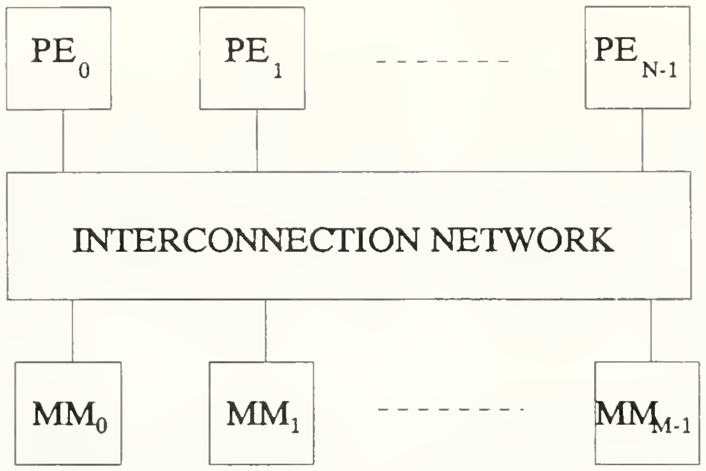

Figure 1. The basic system model

Overall system performance depends on the message throughput that can be achieved by the interconnection network. In practice, message throughput is limited not only by the theoretical bandwidth of the network but by its latency, the time from the generation of a request until a response is received, since a processor may not be able to generate new requests until it has received the response to a previous request.

We are interested in latency and throughput of the class of networks called *delta* networks [13], illustrated in Figure 2. Such networks connect $N=a^n$ PEs to $M=b^n$ MMs via an n-stage network composed of $a \times b$ switches . Much previous work has been done on performance of these networks (see, e.g., [9] [10] [6]). In this paper we will give results for square delta networks, with $M=N$ and $a=b=k$; in particular we will concentrate on *omega* networks (a sub-class of square delta networks with perfect shuffle connections [12]) composed of 2×2 switches.

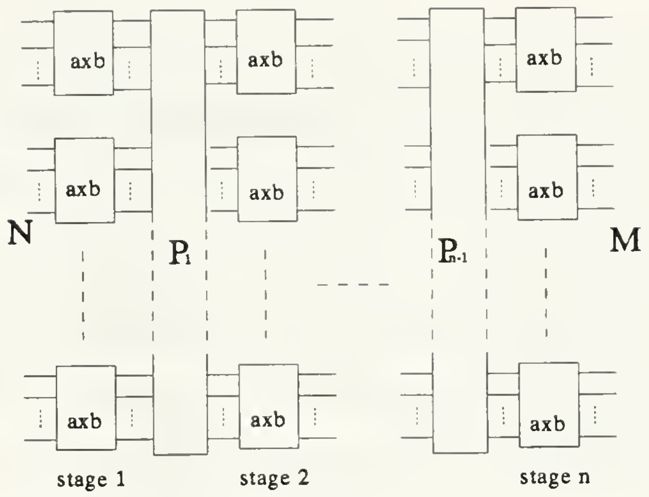

Figure 2. A delta network

The following section describes some possible architectures for switches in the interconnection network and summarizes previous results concerning performance under uniform traffic. Section 3 extends these results for simple "hot spot" traffic. Section 4 discusses simulation methods, including the generation of requests from the PE and responses from the MM; section 5 compares certain architectural features using simulation results.

2. Switch Architectures

Consider a k×k crossbar switching component in a delta network. Its basic function is to forward messages from any of its k inputs to any of its k outputs. It may include buffers to hold messages in case of conflicts for the output ports or blocking from later stages. These buffers may be associated with either input or output ports and may perform extra functions such as combining messages destined for the same memory location or sorting messages according to destination in a later stage.

Each port of the crossbar will accept a certain number of bits per cycle, the packet size. The product of the packet size and crossbar size, k, must be smaller than the number of pins on the package, whether a chip or a board. Thus a design decision to make one large tends to force the other to be small.

2.1. A Taxonomy of Switch Types

Switches can be classified according to the presence or absence of buffers, and according to the location of the buffers. These variations cause differences in performance, for the same cycle time and data path width. The illustrations below show 2×2 switches, for simplicity.

All performance figures in this section are derived from a traffic model in which addresses are uniformly distributed among the MMs, and the interarrival time of requests at the first stage of the network is geometrically distributed. The network is assumed to be an $N \times N$ square delta network, composed of $k \times k$ switches, with $\log_k(N)=n$ stages.

2.1.1. Unbuffered

In this simplest switch design (see Figure 3), a protocol must be used to kill messages in case of conflict. Lost messages must be retransmitted. The probability of an output at a switch in the i^{th} stage is $p_i=1-(1-p_{i-1}/k)^k$, where $p_0=p$ is the offered load on an input port to the network [13]. This can be approximated by

$$p_n=2k/((k-1)n+2k/p) \tag{1}$$

(see [8]). Thus for a square delta network with N PEs and N MMs, the throughput at each output port of the network is limited to $O(1/\log N)$, holding k and p constant. The overall bandwidth of the network is then limited to $O(N/\log N)$.

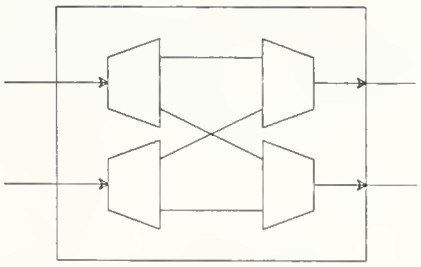

Figure 3. An unbuffered 2×2 switch

The latency of a message, measured from the time a processor makes a request until it is satisfied, is difficult to estimate because of retransmission. Suppose the processor can actually generate requests at a rate b, independent of any responses it receives. Over time, if b is less than the maximum bandwidth of the network, retransmissions will accumulate until the offered load on the network p minus the rate r of rejected messages gives the desired b. At this point, the offered load should stabilize at $p=b+r$. For a given $b=p_n$ equation (1) can be solved for p, b/p will give the probability of a message being accepted, and p/b the average number of trials until it is accepted. If we assume that the transit time in switch cycles of a message accepted by the network is $n+m-1$, where n is the number of stages and m is the number of packets per message, and that a rejected message is retransmitted after twice this amount of time (when no response or a negative acknowledgement is received), then the expected value of the round trip latency is

$$2\times(p/b)\times(n+m-1). \tag{2}$$

The one-way latency may be taken to be half this amount.

2.1.2. k-input buffers, one per output port

For this switch structure, a hardware buffer capable of accepting k inputs in one cycle is needed (see Figure 4). This is the type of switch that has been most thoroughly analyzed in the literature, especially in [9].

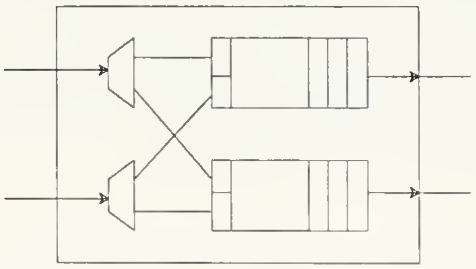
Figure 4. A 2-input buffer 2×2 switch

According to their analysis, if the queues at each switch may grow without bound, ("infinite buffers") then the average switch delay at the first stage is

$$1 + \left(\frac{m^2p(1-1/km)}{2(1-mp)}\right) \tag{3}$$

where m is the number of packets in a message, and the average switch delay at later stages is approximately

$$1 + \left(1 + \frac{4mp}{5k}\right)\left(\frac{m^2p(1-1/k)}{2(1-mp)}\right) \tag{4}$$

The average network traversal time (in one direction) is the sum of the individual stage delays plus the setup time for the pipe, i.e. $(m-1)$.

Note that the network has a capacity of 1/m messages per switch cycle per PE. That is, each PE cannot enter messages at a rate higher than one per *m* cycles, and, conversely, the network can accommodate any traffic below this threshold. Thus, the global bandwidth of the network is theoretically proportional to the number of PEs connected to it. The initial 1 in the expressions for the switch delay corresponds to the time required for a message to be transmitted through a switch without being queued (the switch service time).

These formulas do not take into account the effect of blocking due to full buffers in later stages. Simulations can be used to determine the buffer size necessary to avoid excessive blocking. See Section 5 below.

This configuration, using k-input buffers, has the disadvantage of being somewhat more difficult to realize in hardware than the alternatives using one-input buffers which follow.

2.1.3. One-input buffers, one per input port

In this arrangement (see Figure 5a), outputs of the buffers are multiplexed, and a buffer may be blocked on output by another buffer. Though the simplest type of buffered switch, this is relatively difficult to analyze with conventional methods of queueing theory, due to the blocking which occurs even if "infinite buffers" are assumed.

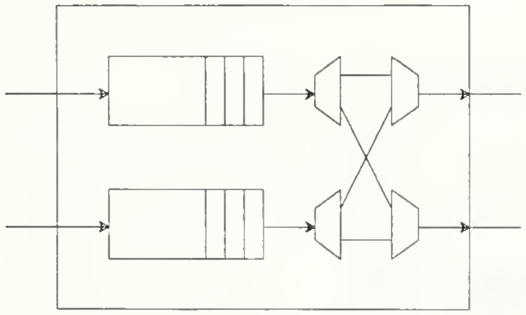

Figure 5a. An output-multiplexed 1-input buffer 2×2 switch

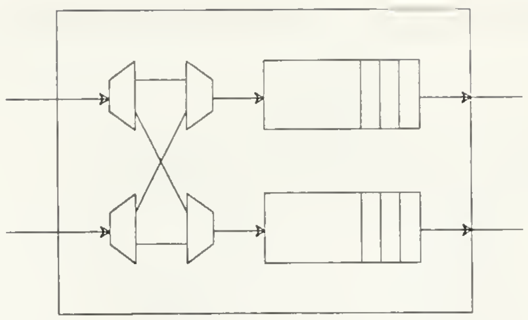
Figure 5b. An input-multiplexed 1-input buffer 2×2 switch

Suppose that α is the probability that a buffer is not empty; then, under the assumption that the queue lengths of the different buffers in a switch are independent, $b=1-(1-\alpha/k)^k$ is the average number of items exiting from each output port each cycle. Since α cannot be greater than one, the maximum bandwidth per port is $1-(1-1/k)^k$. At $k=2$, this is .75; as k gets large, this approaches $1-1/e=.63$.

The average number of items exiting from the switch each cycle is then bk, and the average number of items at the front of buffers ready for service is αk. Let x be the average service time for each item at the front of a buffer. Then by Little's Law $xbk=\alpha k$, so that $x=\alpha/b$. An approximate value for the waiting time at the first stage, assuming a queue with service time x and load p per port would then be

$$w_1=\frac{(1-1/k)xp}{2(1-xp)}.$$

Under light load, when the probability that the buffer is not empty is small, x approaches 1 and the waiting time approaches that of the k-input buffers discussed above. Possibly performance in later stages and for multiple packet messages is likewise analogous.

The above analysis is marred by the assumption that the queue lengths are independent, when in fact service at one buffer depends on the queue length at the others. The actual value of α for a given load p is also difficult to determine, particularly when finite buffer effects are taken into account. An iterative method for determining queue length probabilities in the case of 2×2 switches, assuming independence between the lengths of the two queues, is given in [6]. They get a fit within 10 to 15 percent of simulation values, but their method for network evaluation depends on concatenating a switch model in which packets that arrive when a buffer is full are lost. Table 1 compares bandwidth and latency estimates for 2×2 switches using their method with the estimates from formulas (3) and

(4) in the preceding section. The latency estimates from the iterative method seem low.

Offered Load	Latency	
	One-input buffers	Two-input buffers
0.1	6.002	6.317
0.2	6.015	6.737
0.4	6.186	7.307
0.6	7.433	11.025
0.8	20.357*	20.200

Table 1 Latency predictions for a 64-PE network, single packet messages.

*At this offered load, actual throughput is predicted at .716.

With this simple type of buffer, arranging the multiplexers before the buffers instead of after should make no difference except in the first and final stages (see Figure 5b). Thus in [10] this arrangement is called "buffers between the switches."

2.1.4. One-input buffers, k buffers per output port

This configuration (see Figure 6) is a way to use more instances of a simpler type of buffer to approximate the performance of the k-input buffers discussed above. A packet leaves an output port whenever any of the k associated buffers has data; if more than one has data, arbitration must occur.

The analysis of each single buffer involves a load dependent service time like that for the previous one-input buffer configuration. However, if we look at the k buffers associated with an output port as a single queueing system with a service rate of 1 per cycle, then the expected waiting time in the first stage is the same as that of the k-input buffer. However, the service discipline within the queueing system is no longer first come, first serve, so the variance of the waiting time may differ. This in turn may effect the result at later stages.

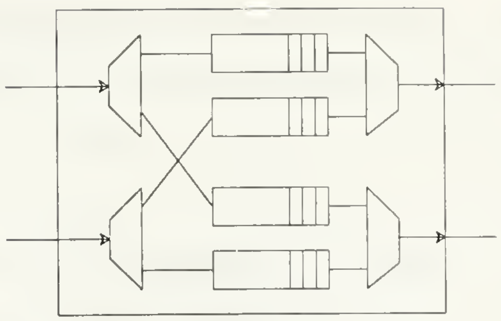
Figure 6. A 4 1-input buffer 2×2 switch

Note that if we were to have a one-input, k-output buffer, this would be essentially the same configuration as k one-input, one output buffers (see Figure 7).

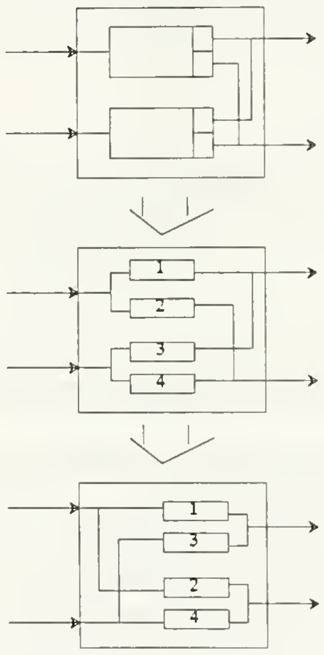

Figure 7. Transformation of 2 one-input, two-output buffers to 4 one-input buffers

In [10], this configuration is called "buffers within the switches" and according to their simulations showed better performance than the "buffers between the switches," especially for high load.

2.2. Increased functionality of buffers

When designing a buffered network, it is tempting to do something with the messages while they are waiting, if so doing either increases the overall throughput or performs some other useful function. We have explored combining messages and multiple clear-to-send signals. Other enhanced buffer functions might include giving priority to loads over stores, particularly in a "store-through" environment, or increasing the kinds of operations which can be combined.

2.2.1. Combining messages

Combining messages when they meet at a switch is one example of such increased functionality. In particular, the Ultracomputer project has proposed combining fetch-and-add operations as well as loads and stores at the switches [1] [5]. The fetch-and-add operation, useful as a synchronization primitive and in many parallel algorithms, is an indivisible add to memory; its format is F&A(X,e), where X is an integer variable and e is an integer expression. The operation is defined to return the (old) value of X and to replace X by the sum X+e.

Concurrent fetch-and-adds must satisfy the *serialization principle* [2]. Fetch-and-add operations simultaneously directed at X cause it to be incremented by the sum of the expressions. Each operation yields the intermediate value of X corresponding to some order of execution.

When two fetch-and-adds referencing the same shared variable, say F&A(X, e) and F&A(X, f), meet at a switch, the switch forms the sum e+f, transmits the combined request F&A(X, e+f), and stores the value e in its local memory. When the value Y is returned to the switch in response to F&A(X, e+f), the switch returns Y to satisfy one request, F&A(X, e), and Y+e to satisfy the other, F&A(X, f). Assuming that the combined request was not further combined with yet another request, memory location X is properly incremented, becoming X+e+f. If other fetch-and-add operations updating X are encountered, the combined requests are themselves combined. The associativity of addition guarantees that this procedure gives a result consistent with the serialization principle. Other associative operations can be combined in a similar manner.

Since combined requests can themselves be combined, any number of concurrent memory references to the same location can be satisfied in the time required for one shared memory access from a single PE. This property permits the bottleneck-free implementation of coordination protocols in which many PE's access the same variables ("hot spots"). Even moderate hotspot traffic can severely degrade all memory access, not just access to shared coordination locations (see [14] and sections below).

2.2.2. Multiple clear-to-send signals

In a network with buffers at the output ports, an input port must block and refuse to accept further messages if any buffer it feeds is full. A "clear-to-send" (CTS) signal from each input port to the output port in the preceding stage must be lowered whenever the port cannot accept a message (see Figure 8).

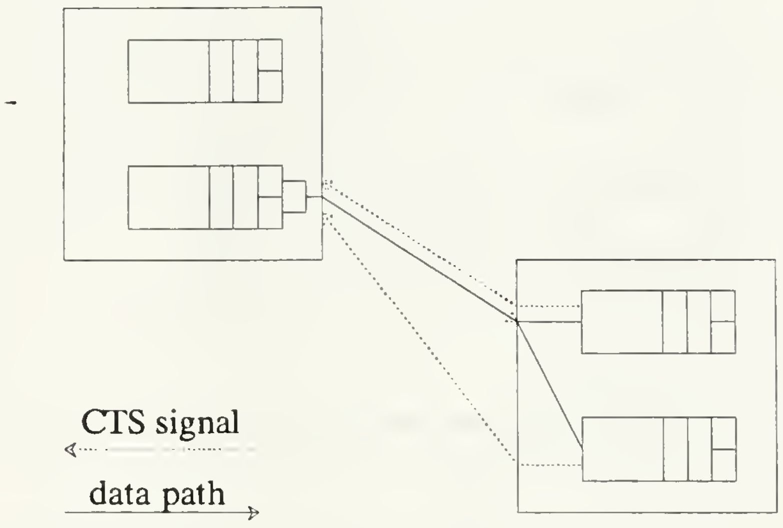

Figure 8. Switches with 2 CTS signals and 2-output buffers

This results in a loss of performance when not all the buffers in the later stage are full, since a message destined for a buffer that has space may nevertheless have to wait. One solution is to have multiple CTS signals, one associated with each of the output ports. Then a message will be blocked only if the output port to which it is destined in the next stage is actually full. The effectiveness of this method is further enhanced if the output buffers at the earlier stage have the capability of choosing the message to be sent according to its output destination in the next stage. This can be done by sorting messages into separate buffers as they arrive or by the use of an associative buffer which selects messages by address bits. Either alternative requires additional hardware complexity.

2.3. Effects of data path width and crossbar size

Increasing the crossbar size seems attractive because it cuts the number of stages in the network, thus decreasing component count and apparently reducing latency. For unbuffered networks, the bandwidth (measured in packets per cycle) also increases. However, for switches with one one-input buffer per output port the maximum bandwidth may go down slightly as crossbar size increases, as discussed in section 2.1.3 above.

If decreasing the width of the data path is done in order to allow packaging of a larger crossbar (as is done in [3]), maximum bandwidth in bits goes down and message latency generally goes up, due to increased pipeline fill time. If the number of packets per message is significant compared to the number of stages in the network, or if increasing the number of packets per message raises the bandwidth close to network capacity, latency will go up substantially as the data path width is decreased. Since decreasing the data path width while increasing the crossbar size saves wire and component cost, such a network could be duplicated to yield a bandwidth comparable to a network with a wider data path. The portion of the delay due to pipeline fill time would still be a problem, however. A substantial decrease in switch cycle time must result from narrowing the data path in order to give a network with comparable performance.

In Table 2, formulas (1), (2), (3) and (4) from section 2.1 are used to compute one-way network latency for varying message throughputs in different switch configurations. Switch configurations grouped together in the table have the same number of pins per package. If the message throughput desired exceeds the maximum bandwidth of that configuration, the latency entry is left blank.

		Latency by throughput								
		Unbuffered				k-input buffers				
Packets/ message	Crossbar size	.05	.1	.2	.4	.05	.1	.2	.4	.8
2	2	8.2	10.0	17.5	-	7.7	8.5	11.4	36.4	-
4	4	7.7	10.9	-	-	7.5	10.1	32.1	-	-
8	8	13.8	-	-	-	14.4	42.0	-	-	-
1	2	6.5	7.1	8.6	15.0	6.1	6.3	6.7	8.1	20.2
2	4	4.5	5.2	7.3	-	4.3	4.7	6.0	16.8	-
4	8	6.1	7.7	-	-	6.0	7.7	21.4	-	-
1	4	3.2	3.4	3.9	5.5	3.1	3.2	3.4	4.0	9.1
2	8	3.3	3.4	4.6	-	3.2	3.5	4.3	11.1	-
1	8	2.1	2.2	2.4	3.1	2.0	2.1	2.4	3.1	5.9

Table 2a Latency in switch cycles for a 64-PE network, different switch configurations

		Latency by throughput							
		Unbuffered			k-input buffers				
Packets/ message	Crossbar size	.05	.1	.2	.05	.1	.2	.4	.8
2	2	18.6	34.5	-	14.4	16.2	22.0	74.1	-
4	4	16.4	-	-	12.1	17.4	62.9	-	-
8	8	-	-	-	22.0	78.6	-	-	-
1	2	14.1	17.1	21.8	12.3	12.7	13.5	16.4	42.0
2	4	9.0	12.7	-	7.7	8.5	11.2	33.7	-
4	8	10.8	-	-	9.0	12.5	40.7	-	-
8	16	22.8	-	-	18.1	59.1	-	-	-
1	4	6.8	7.7	10.9	6.2	6.3	6.7	8.0	19.1
2	8	6.1	7.7	-	5.4	6.0	7.7	21.7	-
4	16	8.35	13.71	-	7.5	10.0	30.5	-	-
1	8	4.4	4.9	6.2	4.1	4.2	4.5	5.3	12.2
2	16	34.6	5.6	9.1	4.3	4.7	6.0	16.2	-
1	16	3.2	3.5	4.2	3.1	3.2	3.4	4.0	9.0

Table 2a Latency in switch cycles for a 64-PE network, different switch configurations

*Throughput is in messages per switch cycle.

3. A model for single "hot spot" traffic

Suppose each PE issues two types of requests: hot spot requests directed to a particular MM and other requests uniformly distributed among all N MMs. Assume the uniform requests are colored *white*, with request rate W; and the hot spot requests are colored *red*, with rate R. Each PE issues $p=W+R$ requests per cycle. The paths of the hot spot requests produce a *red* traffic tree rooted at the hot MM, and spanning all the PEs as leaves.

We label each input and output of a switch with a designation of its traffic rate. The labeling scheme is as follows:

At stage 1: Initially, we label all inputs $p_{0,0}$; the outputs in the red traffic tree are labeled $p_{1,1}$; those not in the red traffic tree $p_{0,1}$.

At stage i, $1<i\leq n$, we have have following two labeling rules:

(1) If all inputs of a switch are labeled $p_{i-1,i-1}$, then the output which leads to the hot MM is labeled $p_{i,i}$; the other k-1 outputs $p_{i-1,i}$. Such a switch will be in the hot traffic

tree.

(2) If all inputs of a switch are labeled $p_{j,i-1}$, $j<i-1$, then all k outputs are labeled $p_{j,i}$.

We start the labeling process according to the above labeling scheme and stop only when there are no labeling rules applicable.

When the labeling process stops there are two possible outcomes: all network links are labeled; or some network links are not labeled. If the labeling process stops without all network links labeled there must be a switch node which has different input labels. Otherwise we could apply either rule (1) or (2) to proceed with the labeling process. On the other hand, if all network links are labeled at the end of the labeling process, all input ports of the same switch node must have the same label.

The following theorem is proven in [7]: Under the above definition, each switch node in a delta network has all its inputs labeled with the same $p_{j,i-1}$ where i is the stage of the node, and $j<i$. Figure 9 shows a labeled network.

Thus under the single "hot spot" traffic model, the analysis of network performance is reduced to two simple cases corresponding to each of the labeling rules: (1) those switches which are located in the hot traffic tree; and (2) those switches which are not in the hot traffic tree. For both cases all inputs of the same switch node have the same label and same history and therefore have the same distribution. The switches in case (2) have the same properties as the switches in the uniform traffic model and can be analyzed with the same methods. For case (1), we have one output port preferred by all the input ports in a switch, which needs special treatment, but presents no fundamental difficulty.

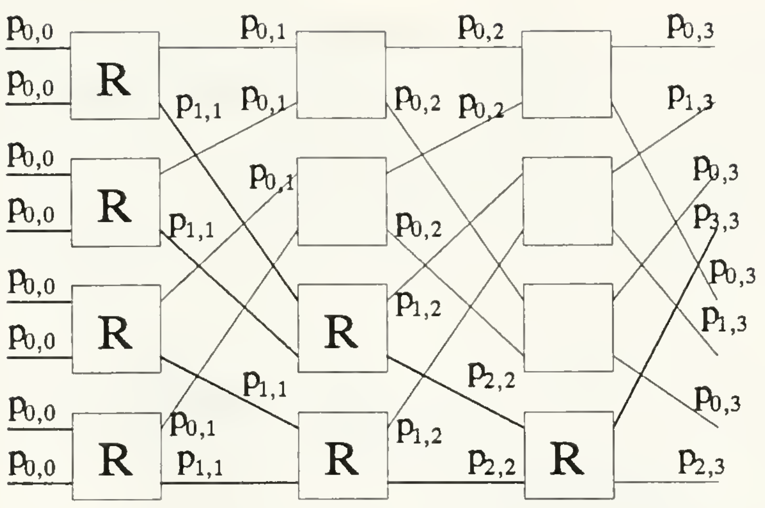

Figure 9. A labeled $2^3 \times 2^3$ delta network with hot location at MM #3

For an unbuffered network, if the switch node is in the red traffic tree, the probability that a "hot" output has a message is:

$$p_{j,j} = 1 - [(1 - r_{j-1} p_{j-1,j-1} - (1 - r_{j-1}) p_{j-1,j-1}/k]^k.$$

and the probability a white output has a message is:

$$p_{j-1,j} = 1 - [1 - (1 - r_{j-1}) p_{j-1,j-1}/k]^k.$$

where,

$$r_j = \frac{Rk^j}{Rk^j + W} \quad j = 0, 1, \ldots, n.$$

For nodes not in the red traffic tree,

$$p_{i,j} = 1 - (1 - p_{i,j-1}/k)^k.$$

Take a $4^5 \times 4^5$ delta network as an example. At the outputs of the last stage, traffic can be divided into 6 classes:

$p_{5,5}$: "hot" traffic, one output only;

$p_{4,5}$: the traffic conflicts with the "hot" traffic for 4 stages, 3 outputs;

$p_{3,5}$: the traffic conflicts with the "hot" traffic for 3 stages, 12 outputs;

$p_{2,5}$: the traffic conflicts with the "hot" traffic for 2 stages, 48 outputs;

$p_{1,5}$: the traffic conflicts with the "hot" traffic for 1 stage, 192 outputs; and

$p_{0,5}$: the traffic has no conflict with the "hot" traffic, 768 outputs.

The effects of "hot" traffic on the other five classes of traffic are shown in Figure 10.

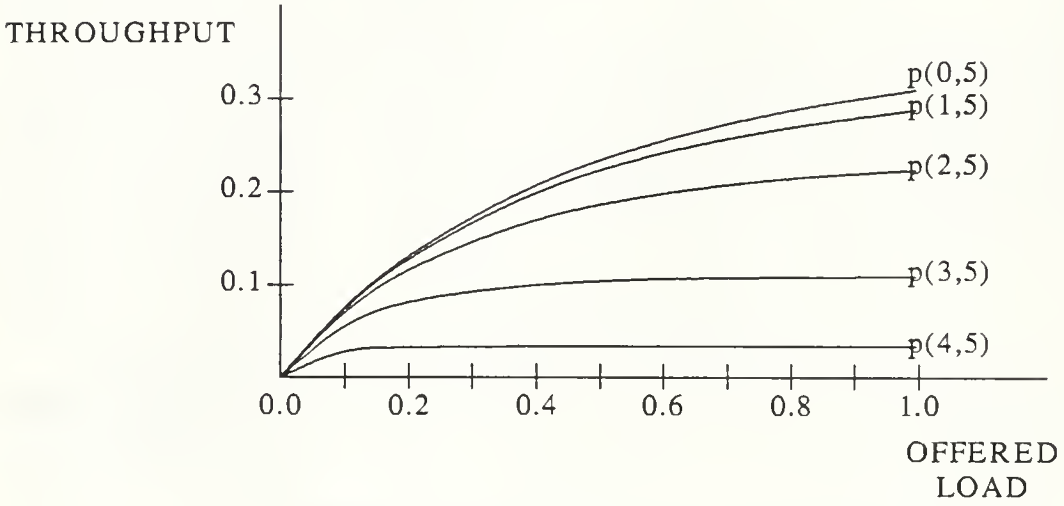

Figure 10a. A $4^5 \times 4^5$ unbuffered delta network under 10% hot spot offered load

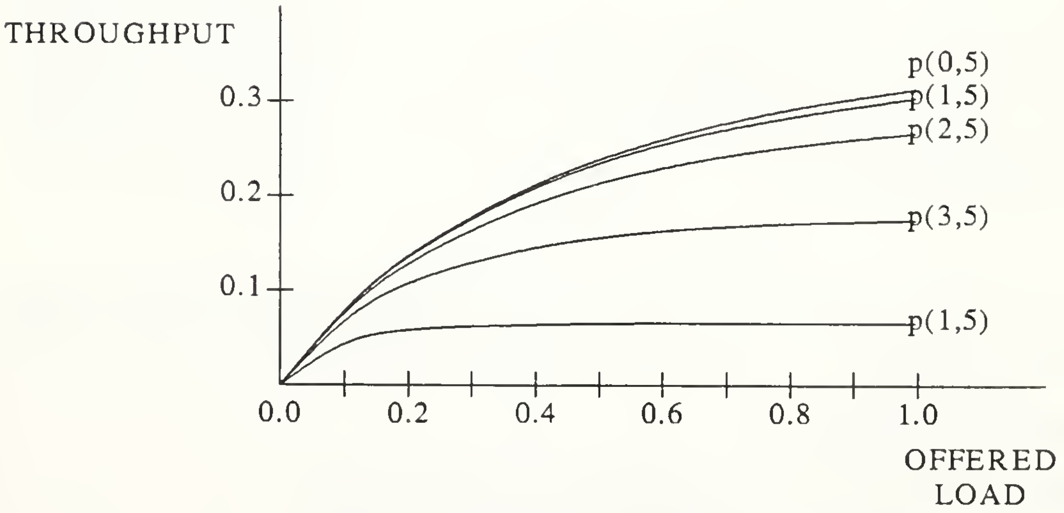

Figure 10b. A $4^5 \times 4^5$ unbuffered delta network under 5% hot spot offered load

For buffered networks, assuming infinite buffers, the network is non-blocking and every request is able to get into the network. If the network has a stable state for a given offered load,

$$p_{i,i} = W + Rk^i, \qquad i = 0, 1, \ldots, n$$

and

$$p_{j,i} = W, \qquad i > j,\ i = 1, 2, \ldots, n.$$

For the network to be stable, we must have

$$W + NR \leq 1.$$

which severely restricts the amount of hotspot traffic which can be handled without combining for large N.

The techniques in [9] may be applied using the above input rates at each stage to get approximate delay information for given hot spot rates.

4. Simulation methods

Our simulator allows the simulation of networks composed of 2×2 switches under various assumptions of switch architecture, PE request generation and memory behavior. Output statistics include the average latency of a message, the average bandwidth in messages, average queue length per stage and number of combines that occur.

Switches with 2 two-input buffers, 4 one-input buffers and 2 one-input buffers can all be simulated. Buffer size at switches can be varied. Different clear-to-send protocols can be tested, and different combining alternatives corresponding to different hardware implementations can be specified.

PE request generation can be done in two ways: (1) generate a request with some fixed probability whenever permitted by the first stage of the network or by the capacity of a finite PE request queue; (2) generate a request with some fixed probability whenever a PE "wants to," which requires the simulation of an infinite request queue at the PE. In the first case, the offered load is the probability of generating a request on any cycle when the PE is not blocked. The effective throughput will be less than the offered load. In the second case, messages accumulate latency in the request queue at the PE, but as long as that queue reaches a steady state, the effective throughput will be equal to the offered load.

We simulate an infinite request queue by keeping a counter t_0 which represents the arrival time at the queue of the last request removed from the queue. Then, since we assume a memoryless system with a constant birth rate, to get t_1, the arrival time at the queue of the next request taken from the queue, we use the distribution:

$$\mathrm{Prob}(t_1=t) = p(1-p)^{t_1-t_0-1}$$

Our results show very little difference between the two methods of PE request generation, except when the load is heavy. In that case the delay as computed with an infinite request queue is greater (see Figure 11). Also, under lighter load but with hotspot requests to induce congestion, an infinite PE request queue with an offered load of 30 percent will show better performance with larger buffer size in the switches, but a finite request queue, operating at maximum offered load but achieving only 30 percent throughput, will experience larger delays with larger buffer sizes by pushing requests into the network until the buffers are full, and then becoming blocked (see Figure 12).

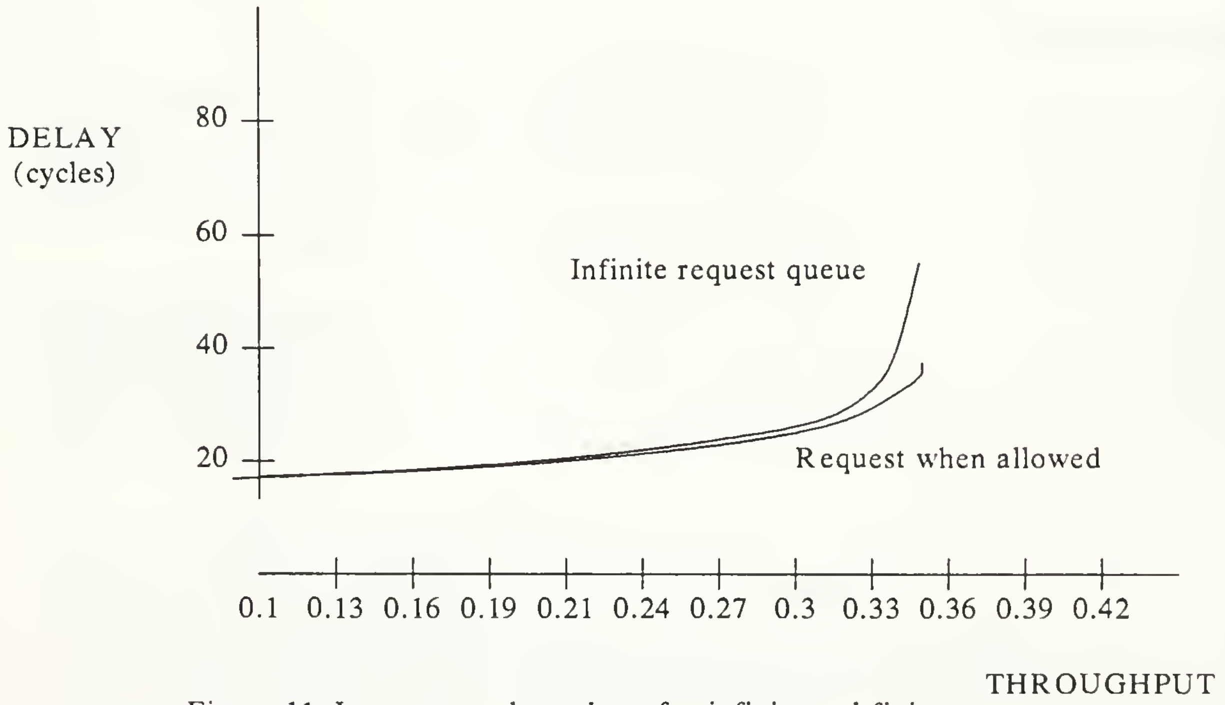

Figure 11. Latency vs. throughput for infinite and finite request queues

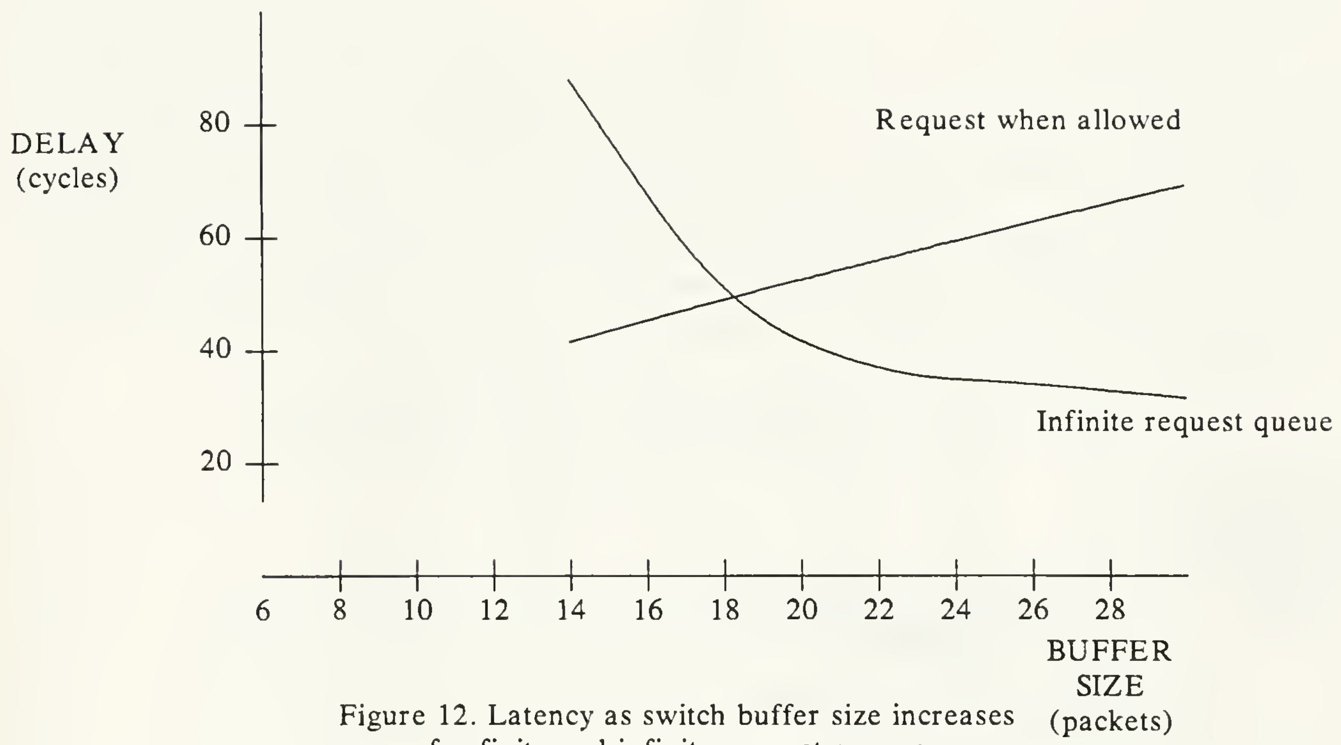

Figure 12. Latency as switch buffer size increases for finite and infinite request queues.

PE request generation can also simulate a processor which is allowed only a small number of requests outstanding before it must quit generating requests.

Memory behavior is modeled with both a cycle and an access time. The cycle time is used to determine whether or not the memory will accept a message; the access time is used in computing total latency from processor to memory. Cycle and access times may vary for loads, stores and other operations. The intervention of the memory makes delays on the forward and return paths not always symmetric, and must be included for accurate system modeling.

5. Simulation results

Our simulation results are carried out with parameters that we consider reasonable for a current system. Thus we simulate two packet messages, operations of 60 percent loads and 40 percent stores, 2 switch cycle memory access and cycle time for loads, and access time of 0 and cycle time of 2 for stores. We use a finite PE request queue (in the simulations below, of size 0) as a more realistic model, since actual processors do not have the capability to continue generating requests when blocked. All results given here are for a 64

PE omega network composed of 2×2 switches.

Figure 13 a-f shows latency and throughput under different offered loads with different sizes for the three buffered switch types: two-input buffers, one at each output; one-input buffers, 2 at each output; and one-input buffers, one at each input. Since two-packet messages are simulated, the maximum throughput that can be achieved is 50 percent, and the best possible latency for loads is twice the number of stages (12) plus a cycle for pipeline fill at PE and MM (2) plus memory access time (2) for a total of 16.

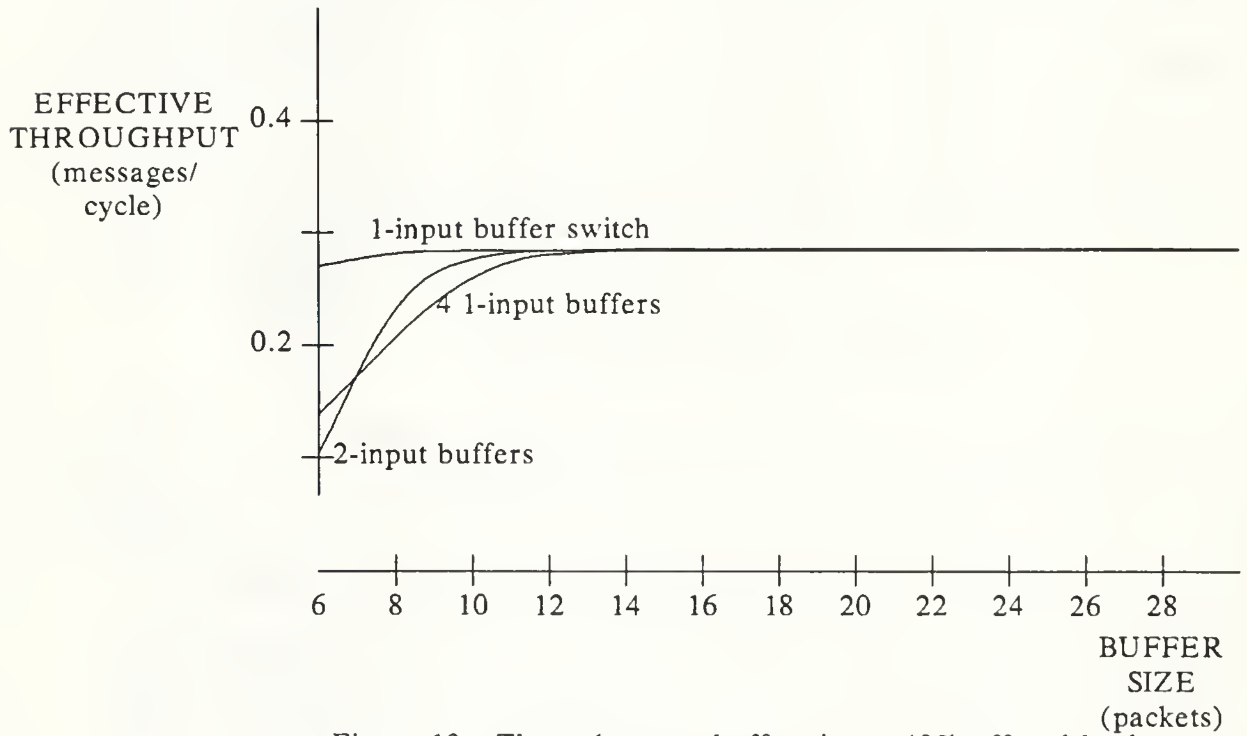

Figure 13a. Throughput vs. buffer size at 40% offered load

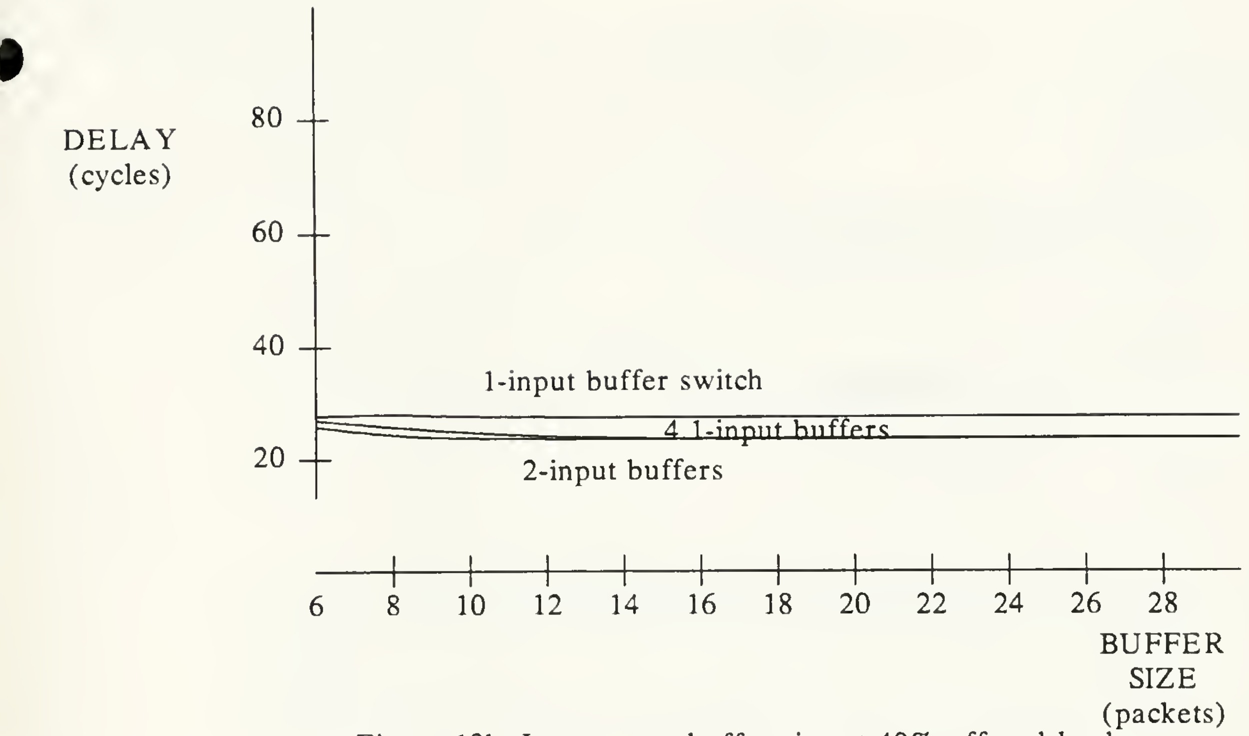

Figure 13b. Latency vs. buffer size at 40% offered load

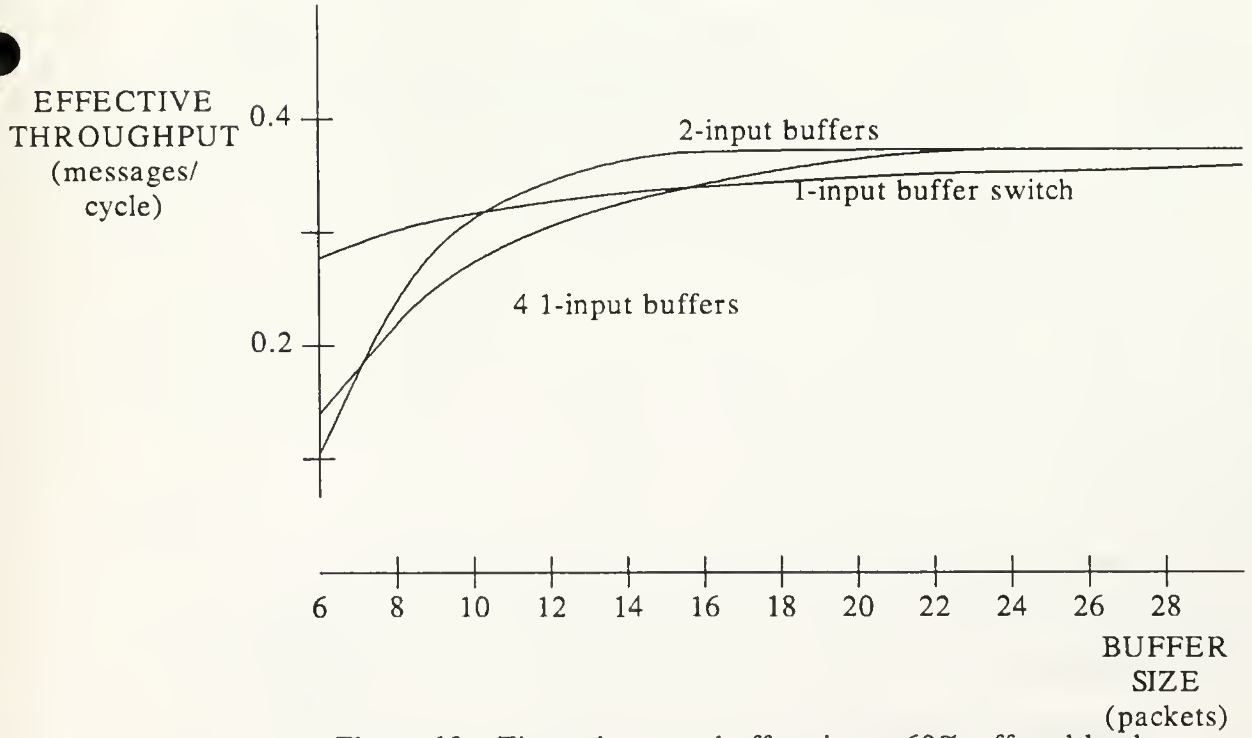

Figure 13c. Throughput vs. buffer size at 60% offered load

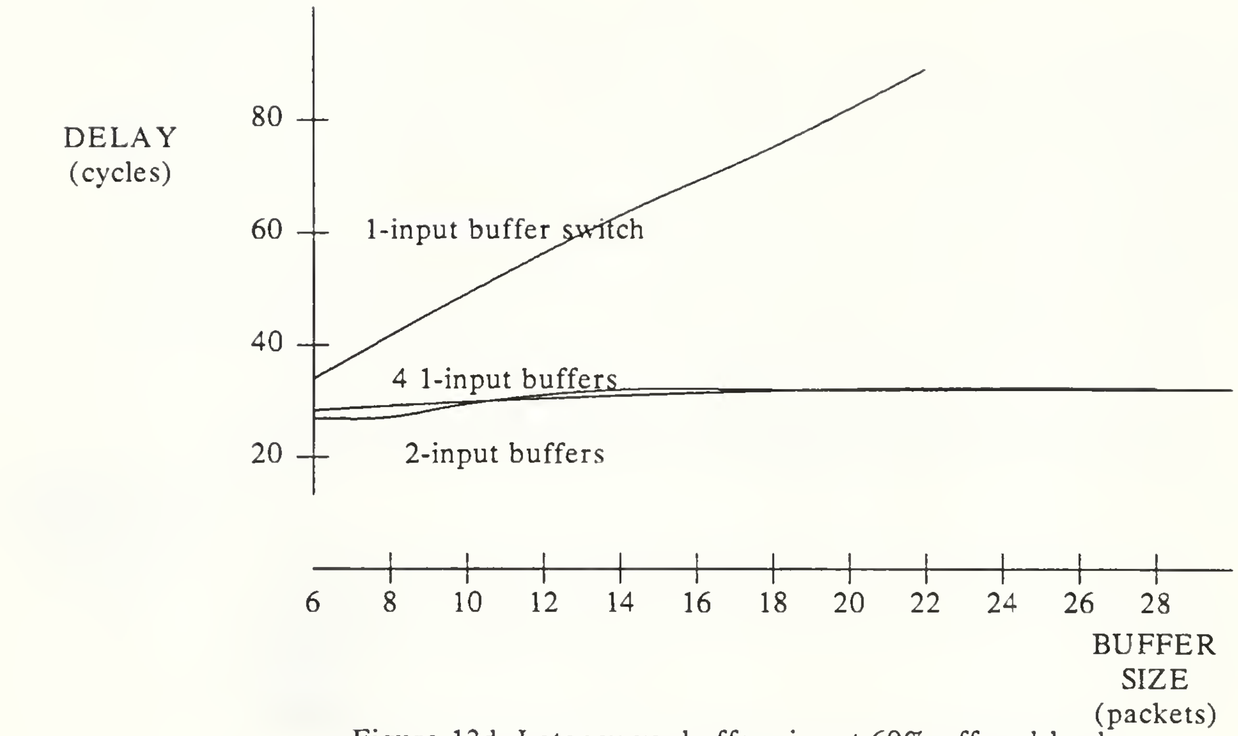

Figure 13d. Latency vs. buffer size at 60% offered load

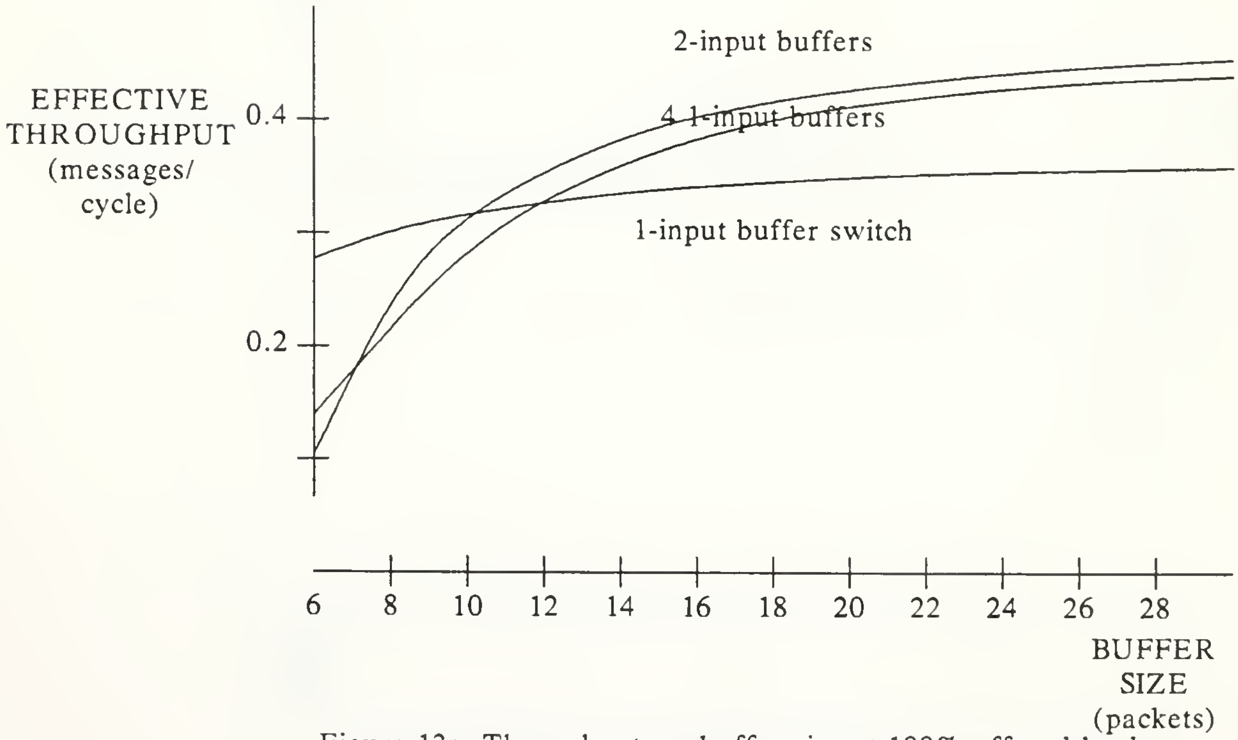

Figure 13e. Throughput vs. buffer size at 100% offered load

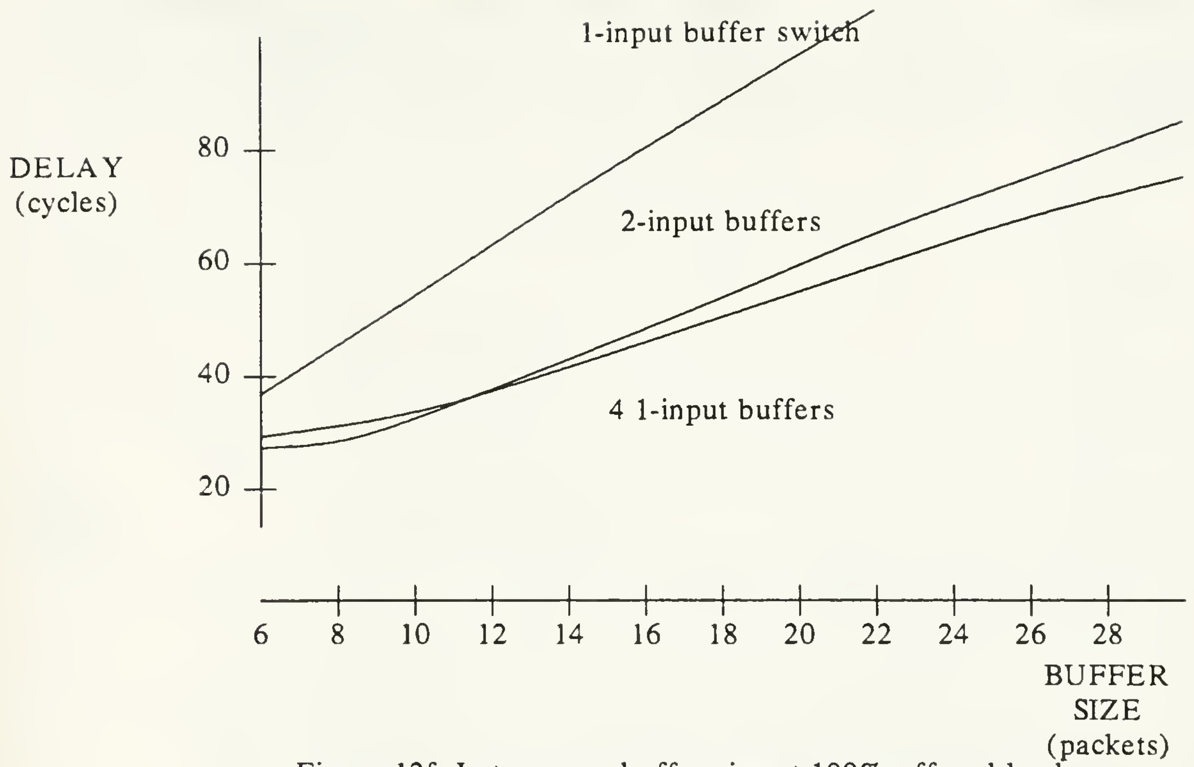

Figure 13f. Latency vs. buffer size at 100% offered load

The buffer size given for the one-input buffers, two at each output, is twice the size of a single buffer, to give a fair comparison in terms of total amount of buffer space. The protocol governing the output-buffering switches requires that two message slots be reserved so that a switch can still accept two messages on its inputs the cycle immediately after it has been blocked by a later stage. Therefore the minimum useful buffer size is 6 packets, and at the small buffer sizes of 6 or 8 packets the output-buffering schemes show poorer performance than the input-buffering scheme. However, for larger buffer sizes, especially at heavier loads, the output-buffering schemes can achieve a higher throughput.

For all these switch types, increasing the buffer size beyond a certain point results in negligible gain in maximum throughput at a severe cost in latency. In actual systems, this effect may be self-correcting, since higher latencies will tend to lower the request rate from the processor. A system imposed limit on the PE's maximum request rate might also be desirable, to avoid loading the network above the capacity at which it functions well.

Figure 14 shows the latency of different buffered switch types at different throughputs, operating with buffer size large enough to get good performance. For the light loads, there is little difference. However, even in a system where the average load is expected to be light, the switch types with better performance at heavier loads may make the network more robust to bursty traffic.

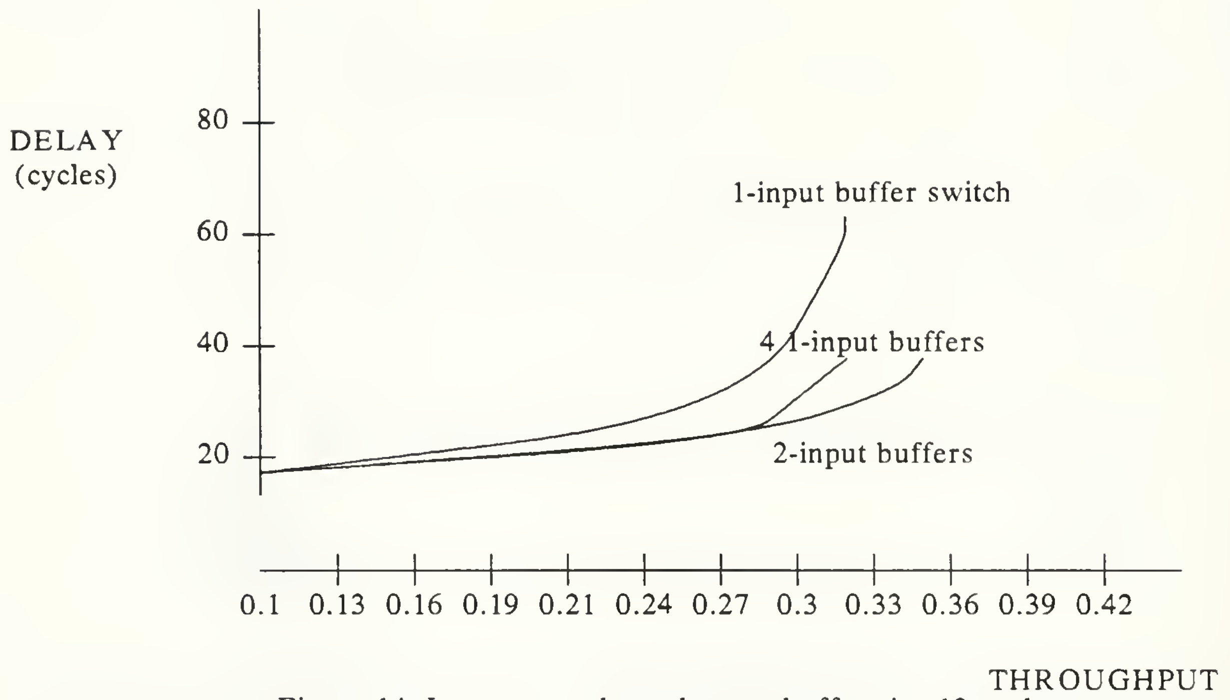

Figure 14. Latency vs. throughput at buffer size 12 packets

Table 3 demonstrates performance improvement due to combining. These simulations were conducted under maximum load, with 2-input buffers and a buffer size of 18 packets. The number of combines is given as a proportion of total memory requests. A surprisingly small number of combines can make a significant improvement in performance.

Figure 15 compares the performance of our standard protocol with one CTS signal to 2 CTS signals with output buffers able to forward the first message that is not blocked and to 2 CTS signals with simple output buffers. Without the improved buffer, performance enhancement is small; even with the improved buffer the effect is noticeable only at heavy loads.

Hot-spot-rate	Throughput		Latency		Proportion of combines
	No combining	Combining	No combining	Combining	
1%	30%	40%	49	50	.42%
5%	12%	34%	100	47	.50%

Table 3. Performance improvement due to combining.

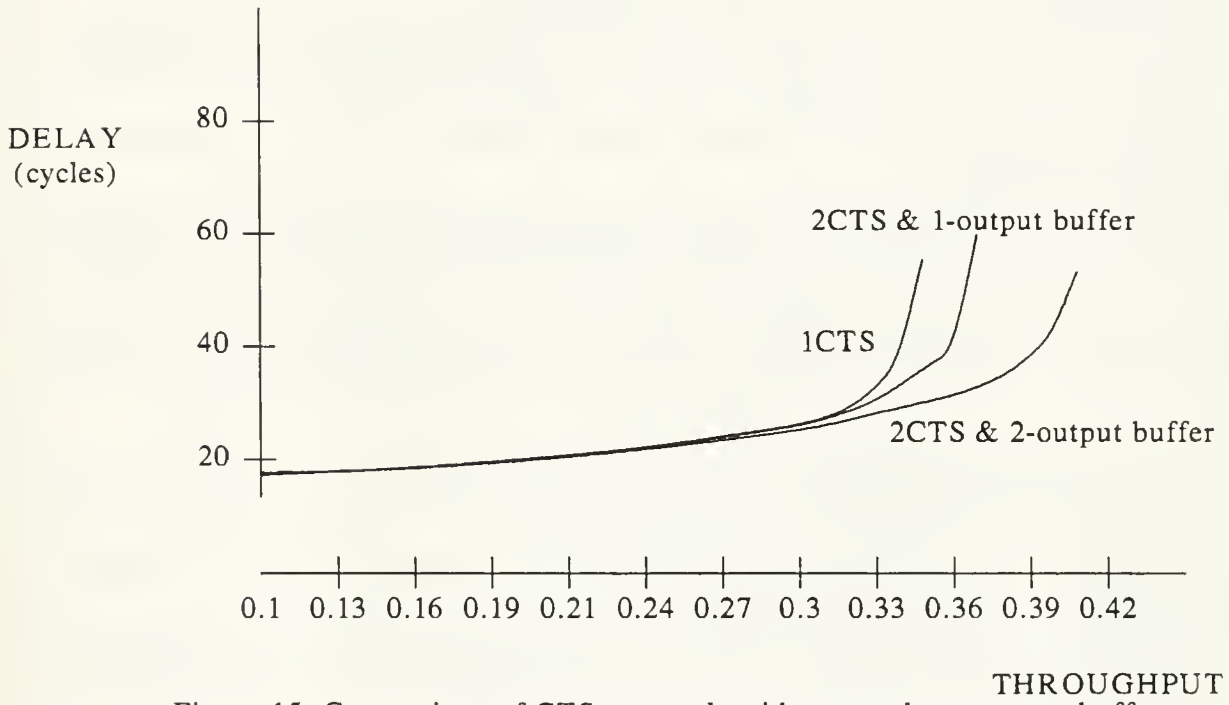

Figure 15. Comparison of CTS protocols with one and two output buffers.

6. Further work

We are continuing work on analytic solutions, particularly on the joint distribution of queue lengths for buffers within a switch, on the output distribution, and on finite buffers. We are interested in developing analytic models that include the effect of blocking at memory on the round trip performance. Better analytic solutions are needed to judge initial proposals for network design; simulations can be most usefully conducted only after a design has been elaborated.

We are also gathering traces from parallel programs to be used in conducting trace-driven simulations of the network. Our goal is to integrate the network simulator into a system simulation environment that can simulate increasingly complex models of processor and memory, as well as switch behavior.

References

[1] S. Dickey, R. Kenner, M. Snir and J. Solworth, "A VLSI Combining Network for the NYU Ultracomputer," *Proceedings of the International Conference on Computer Design*, pp. 110 - 113, October 7, 1985.

[2] K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, " The Notion of Consistency and Predicate Locks in a Database System," *CACM*, pp. 624-633, November 1976.

[3] M. A. Franklin and S. Dhar, "Interconnection Networks: Physical Design and Performance Analysis," *Journal of Parallel and Distributed Computing*, pp. 352 - 372, September, 1986.

[4] A. Gottlieb, "An Overview of the Ultracomputer Project," Ultracomputer Note #100, Courant Institute, New York University, 1986.

[5] A. Gottlieb, B.D. Lubachevsky, and L. Rudolph, "Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors", *ACM TOPLAS* **5**, pp. 164-189, Apr. 1983.

[6] D. T. Harper III and J.R. Jump, "Performance Evaluation of Reduced Bandwidth Multistage Interconnection Networks," *The 14th Annual International Symposium on Computer Architecture*, May 1987, pp. 171-145.

[7] Y. S. Liu, "Delta Network Performance and 'Hot Spot' Traffic," Ultracomputer Note #132, Courant Institute, New York University, 1988.

[8] C.P. Kruskal and M. Snir, "The Performance of Multistage Interconnection Networks for Multiprocessors", *IEEE Trans. Comp.* C-32, pp. 1091-1098, 1983.

[9] C. P. Kruskal, M. Snir and A. Weiss, "The Distribution of Waiting Times in Clocked Multistage Interconnection Networks," *Proc. 1986 Intern. Conf. on Parallel Processing*.

[10] M. Kumar and J. R. Jump, "Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links," *Journal of Parallel and Distributed Computing*, Vol. 1, No. 1, August 1984, pp. 81-103.

[11] M. Kumar and G. F. Pfister, "The Onset of Hot Spot Contention," *Proc. 1986 Intern. Conf. on Parallel Processing*, pp 28-34.

[12] D.H. Lawrie, "Access and Alignment of Data in an Array Processor", *IEEE Trans. Comp.*, C-24 , pp. 1145-1155, Dec. 1975.

[13] J.H. Patel, "Performance of processor-memory interconnection networks for multiprocessors," *IEEE Trans. Comp.*, C-30, pp. 771-780, Oct. 1981.

[14] G.F. Pfister and V.A. Norton, " 'Hot Spot' Contention and Combining in Multistage Interconnection Networks", *Proc. 1985 Intern. Conf. on Parallel Processing*, pp. 790 - 797.

[15] G. F. Pfister et. al, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proc. 1985 Intern. Conf. on Parallel Processing*, pp. 764-771.

[16] R. Rettberg and R. Thomas, "Contention Is No Obstacle to Shared-Memory Multiprocessing", *CACM*, Volume 29, Number 12, December 1986, pp. 1202-1212.

DATE DUE	BORROWER'S NAME
